天翼云媒体存储

Python SDK 使用指导书

2026-02-26

天翼云科技有限公司

天翼云媒体存储python SDK使用指导书

SDK安装

安装PIP

如果你还没有安装pip命令，以Centos为例，请使用以下命令联网安装：

```
yum -y install epel-release
yum install python-pip
```

安装SDK

1、下载xos-python-sdk.zip并解压进入，下载地址为： xos-python-sdk.zip

```
unzip xos-python-sdk.zip && cd xos-python-sdk
```

2、使用pip命令安装

```
pip install --no-index --find-links=. boto3
```

初始化

使用说明

使用sdk之前必须先初始化sdk，新建s3_client，通过s3_client使用媒体存储功能。

代码示例

```
import botocore.config
import botocore.session

class S3Demo(object):
    def __init__(self):
        config = botocore.config.Config(
            # signature_version='s3v4',  # s3 or s3v4, 签名类型
            # s3={'addressing_style': 'virtual'},     # virtual|path, default
virtual
            # connect_timeout=60,                     # default 60 seconds
            # read_timeout=60,                        # default 60 seconds
        )

        session = botocore.session.get_session()
        self.s3_client = session.create_client(
            's3',
            aws_access_key_id='<your-access-key>',
            aws_secret_access_key='<your-secret-key>',
```

```
            endpoint_url='<your-endpoint>', # e.g. http://endpoint or
https://endpoint
            config=config)
```

请求参数

参数	说明
aws_access_key_id	用户账号 access key
aws_secret_access_key	用户账号 secret key
endpoint_url	天翼云资源池的地址，必须指定http或https前缀
config	客户端配置，可以配置连接超时时间

桶相关接口

创建桶

功能说明

您可以通过使用create_bucket创建存储桶。

代码示例

```
def create_bucket(self):
    resp = self.s3_client.create_bucket(
        Bucket='<new-bucket-name>'
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	String	桶名称	是

返回结果

根据返回码判断是否创建成功，200表示成功。

获取桶列表

功能说明

您可以通过使用list_buckets获取桶列表。

代码示例

```
def list_buckets(self):
    print('list_buckets')
    response = self.s3_client.list_buckets()
    for bucket in response['Buckets']:
        print(bucket["Name"])
```

请求参数

无

返回结果

参数	类型	说明
Buckets	Bucket数组	桶列表

判断桶是否存在

功能说明

您可以使用head_bucket接口判断桶是否存在。

代码示例

```
def head_bucket(self):
    resp = self.s3_client.head_bucket(
        Bucket='<your-bucket-name>'
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

根据返回码判断桶是否存在，200表示存在，404表示不存在

删除桶

功能说明

您可以使用delete_bucket删除存储桶。

代码示例

```
def delete_bucket(self):
    resp = self.s3_client.delete_bucket(
        Bucket='<your-bucket-name>'
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

根据返回码判断是否删除成功，204表示删除成功

设置桶访问权限

功能说明

媒体存储支持一组预先定义的授权，称为Canned ACL。每个Canned ACL都有一组预定义的被授权者和权限，下表列出了相关的预定义授权含义。

ACL	权限	描述
private	私有读写	存储桶拥有者有读写权限，其他用户没有访问权限
public-read	公共读私有写	存储桶拥有者有读写权限，其他用户只有该存储桶的读权限
public-read-write	公共读写	所有用户都有该存储桶的读写权限
authenticated-read	注册用户可读	存储桶拥有者有读写权限，注册用户具有该存储桶的读限

您可以通过put_bucket_acl接口设置一个存储桶的访问权限。用户在设置存储桶的ACL之前需要具备WRITE_ACP 权限。

代码示例

```
def put_bucket_acl(self):
    resp = self.s3_client.put_bucket_acl(
        Bucket='<your-bucket-name>',
        ACL='private' # private | public-read | public-read-write |
authenticated-read
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
ACL	string	acl值	是

返回结果

根据返回码判断是否设置成功，200表示成功。

获取桶访问权限

功能说明

您可以通过get_bucket_acl接口获取存储桶的access control list（ACL）信息。存储桶的ACL可以在创建的时候设置并且通过API查看，用户需要具有READ_ACP（读取存储桶 ACL信息）权限才可以查询存储桶的ACL信息。

代码示例

```
def get_bucket_acl(self):
    resp = self.s3_client.get_bucket_acl(
        Bucket='<your-bucket-name>',
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

参数	类型	说明
Owner	Owner	所有者信息
Grants	Grants	每种类型用户的详细权限信息

设置桶策略

功能说明

存储桶授权策略（bukcet policy）可以灵活地配置用户各种操作和访问资源的权限。访问控制列表（access control lists，ACL）只能对单一对象设置权限，而存储桶授权策略可以基于各种条件对一个桶内的全部或者一组对象配置权限。桶的拥有者拥有PutBucketPolicy操作的权限，如果桶已经被设置了policy，则新的policy会覆盖原有的policy。您可以通过put_bucket_policy接口设置桶策略，描述桶策略的信息以SON格式的字符串形式通过Policy参数传入。policy的示例如下：

```
{
    "Id": "<your-policy-id>",
```

```
    "Version": "2012-10-17",
    "Statement" : [{
        "Sid": "<your-statement-id>",
        "Principal": {
            "AWS":["arn:aws:iam:::user/<your-user-name>"]
        },
        "Effect": "Allow",
        "Action": [
            "s3:ListBucket",
            "s3:CreateBucket"
        ],
        "Resource": [
            "arn:aws:iam:::<your-bucket-name>/*"
        ],
        "Condition": "<some-conditions>"
    }]
}
```

元素	描述	是否必要
Id	policy id，可选关键字	否
Version	policy 版本号，固定填2012-10-17	是

Statement的内容说明如下：

元素	描述	是否必要
Sid	statement Id，可选关键字，描述statement的字符串	否
Principal	可选关键字，被授权人，指定本条statement权限针对的Domain以及User，支持通配符“*”，表示所有用户（匿名用户）。当对Domain下所有用户授权时，Principal格式为arn:aws:iam:::user/*。当对某个User进行授权时，Principal格式为arn:aws:iam:::user/<your-user-name>	可选，Principal与NotPrincipal选其一
NotPrincipal	可选关键字，不被授权人，statement匹配除此之外的其他人。取值同Principal	可选，NotPrincipal与Principal选其一
Action	可选关键字，指定本条statement作用的操作，Action字段为媒体存储支持的所有操作集合，以字符串形式表示，不区分大小写。支持通配符“*”，表示该资源能进行的所有操作。例如：“Action”:[“s3:List*”, “s3:Get*”]	可选，Action与NotAction选其一
NotAction	可选关键字，指定一组操作，statement匹配除该组操作之外的其他操作。 取值同Action	可选，NotAction与Action选其一

元素	描述	是否必要
Effect	必选关键字，指定本条statement的权限是允许还是拒绝，Effect的值必须为Allow或者Deny	必选
Resource	可选关键字，指定statement起作用的一组资源，支持通配符“*”，表示所有资源	可选，Resource与NotResource选其一
NotResource	可选关键字，指定一组资源，statement匹配除该组资源之外的其他资源。 取值同Resource	可选，NotResource与Resource选其一
Condition	可选关键字，本条statement生效的条件	可选

代码示例

```
def put_bucket_policy(self):
    resp = self.s3_client.put_bucket_policy(
        Bucket='<your-bucket-name>',
        Policy='{"Version":"2012-10-17","Statement":
[{"Sid":"123","Effect":"Allow",' +
        '"Principal":{"AWS":"*"},"Action":["s3:PutObject","s3:GetObject"], ' +
        '"Resource":["arn:aws:s3:::' + '<your-bucket-name>' + '/*"]}]}'
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Policy	string	策略内容，json字符串	是

返回结果

根据返回码判断是否设置成功，200表示成功。

获取桶策略

功能说明

您可以通过get_bucket_policy接口获取指定存储桶的授权策略。如果您使用的身份不是该存储桶的拥有者，则调用身份必须对指定存储桶具有GetBucketPolicy权限，且属于该存储桶所有者的账户。如果您没有GetBucketPolicy权限，方法将返回403 Access Denied错误。如果您具有正确的权限，但您没有使用属于存储桶所有者账户的身份，则返回405 Method Not Allowed错误。

代码示例

```
def get_bucket_policy(self):
    resp = self.s3_client.get_bucket_policy(
        Bucket='<your-bucket-name>',
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

参数	类型	说明
Policy	string	策略内容，json字符串

删除桶策略

功能说明

您可以通过delete_bucket_policy接口删除指定存储桶的授权策略。如果您使用的身份不是该存储桶的拥有者，则调用身份必须对指定存储桶具有DeleteBucketPolicy权限，且属于该存储桶所有者的帐户。如果您没有DeleteBucketPolicy权限，方法将返回403 Access Denied错误。如果您具有正确的权限，但您没有使用属于存储桶所有者账户的身份，则返回405 Method Not Allowed错误。

代码示例

```
def delete_bucket_policy(self):
    resp = self.s3_client.delete_bucket_policy(
        Bucket='<your-bucket-name>',
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

根据返回码判断是否删除成功，200表示删除成功

设置桶生命周期配置

功能说明

生命周期管理可以通过设置规则实现自动清理过期的对象，优化存储空间。您可以使用put_bucket_lifecycle_configuration接口设置桶的生命周期配置，配置规则可以通过匹配对象key前缀、标签的方法设置当前版本或者历史版本对象的过期时间，对象过期后会被自动删除。

```
def put_bucket_lifecycle_configuration(self):
    resp = self.s3_client.put_bucket_lifecycle_configuration(
        Bucket='<your-bucket-name>',
        LifecycleConfiguration={
            'Rules': [
                {
                    'ID': 'TestOnly',
                    'Expiration': {
                        'Days': 3650,
                    },
                    'Status': 'Enabled',
                    'Filter': {
                        'Prefix': 'documents/',
                    },
                    'Transitions': [
                        {
                            'Days': 365,
                            'StorageClass': 'GLACIER',
                        },
                    ],
                    'AbortIncompleteMultipartUpload': {
                        'DaysAfterInitiation': 123
                    }
                },
            ],
        },
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
LifecycleConfiguration	LifecycleConfiguration	封装了生命周期规则的数组，最多包含1000条规则	是

关于生命周期规则Rule一些说明

参数	类型	说明	是否必要
ID	string	规则ID	否
Status	string	是否启用规则（Enabled|Disabled）	是
Expiration	Expiration	文件过期时间	否
AbortIncompleteMultipartUpload	AbortIncompleteMultipartUpload	未完成上传的分片过期时间	否
Transitions	Transition数组	文件转换到低频存储规则（距离修改时间）	否
Filter	Filter	应用范围，可以指定前缀或对象标签	否

关于Expiration的说明：

参数	类型	说明
Days	int	过期天数

关于AbortIncompleteMultipartUpload的说明：

参数	类型	说明
DaysAfterInitiation	int	过期天数

关于Transition的说明：

参数	类型	说明
Days	int	转换过期天数
StorageClass	StorageClass	要转换的存储类型

关于Filter的说明：

参数	类型	说明
Prefix	string	需要过滤的前缀

返回结果

根据返回码判断是否设置成功，200表示成功。

获取桶生命周期配置

功能说明

您可以使用get_bucket_lifecycle_configuration接口获取桶的生命周期配置。

代码示例

```
def get_bucket_lifecycle_configuration(self):
    resp = self.s3_client.get_bucket_lifecycle_configuration(
        Bucket='<your-bucket-name>'
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

参数	类型	说明
Rules	Rule数组	一个描述生命周期管理的规则数组，一条规则包含了规则ID、匹配的对象key前缀、匹配的对象标签信息、当前版本对象过期时间、历史版本对象过期时间和是否生效标识等信息

关于生命周期规则Rule一些说明

参数	类型	说明
ID	string	规则ID
Status	string	是否启用规则（Enabled|Disabled）
Expiration	Expiration	文件过期时间
AbortIncompleteMultipartUpload	AbortIncompleteMultipartUpload	未完成上传的分片过期时间
Transitions	Transition数组	文件转换到低频存储规则（距离修改时间）
Filter	Filter	应用范围，可以指定前缀或对象标签

关于Expiration的说明：

参数	类型	说明
Days	int	过期天数，默认-1，表示不过期

关于AbortIncompleteMultipartUpload的说明：

参数	类型	说明
DaysAfterInitiation	int	过期天数

关于Transition的说明：

参数	类型	说明
Days	int	转换过期天数
StorageClass	StorageClass	要转换的存储类型

关于Filter的说明：

参数	类型	说明
Prefix	string	需要过滤的前缀

删除桶生命周期配置

功能说明

您可以使用delete_bucket_lifecycle接口删除桶的生命周期配置。

代码示例

```
def delete_bucket_lifecycle(self):
    resp = self.s3_client.delete_bucket_lifecycle(
        Bucket='<your-bucket-name>'
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

根据返回码判断是否设置成功，204表示成功。

设置桶跨域访问配置

功能说明

跨域资源共享 (CORS) 定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。您可以通过put_bucket_cors接口设置桶的跨域访问配置。

代码示例

```
def put_bucket_cors(self):
    resp = self.s3_client.put_bucket_cors(
        Bucket='<your-bucket-name>',
        CORSConfiguration={
            'CORSRules': [{
                'AllowedHeaders': ["*"],
```

```
                'AllowedMethods': ["POST", "GET", "PUT", "DELETE", "HEAD"],
                'AllowedOrigins': ["*"], # 可以使用http://domain:port
                'ExposeHeaders': ["ETag"],
                'MaxAgeSeconds': 3600,
            }],
        },
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
CORSConfiguration	CORSRule数组	跨域访问规则	是

关于跨域访问配置CORSRule的一些说明

参数	说明
AllowedMethods	允许的请求方法
AllowedOrigins	允许的请求源
AllowedHeaders	允许的请求头
ExposeHeaders	允许返回的Response Header
MaxAgeSeconds	跨域请求结果的缓存时间

返回结果

根据返回码判断是否设置成功，200表示成功。

获取桶跨域访问配置

功能说明

跨域资源共享 (CORS) 定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。您可以通过get_bucket_cors接口获取桶跨域访问配置。

代码示例

```
def get_bucket_cors(self):
    resp = self.s3_client.get_bucket_cors(
        Bucket='<your-bucket-name>',
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

参数	类型	说明
CORSRules	CORSRule数组	跨域访问规则

关于跨域访问配置CORSRules的一些说明

参数	说明
AllowedMethods	允许的请求方法
AllowedOrigins	允许的请求源
AllowedHeaders	允许的请求头
ExposeHeaders	允许返回的Response Header
MaxAgeSeconds	跨域请求结果的缓存时间

删除桶跨域访问配置

功能说明

跨域资源共享 (CORS) 定义了在一个域中加载的客户端 Web 应用程序与另一个域中的资源交互的方式。利用 CORS 支持，您可以构建丰富的客户端 Web 应用程序，同时可以选择性地允许跨源访问您的资源。您可以通过delete_bucket_cors接口删除桶跨域访问配置。

代码示例

```
def delete_bucket_cors(self):
    resp = self.s3_client.delete_bucket_cors(
        Bucket='<your-bucket-name>',
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

根据返回码判断是否设置成功，204表示成功。

设置桶版本控制状态

功能说明

您可以通过put_bucket_versioning设置存储桶版本控制状态。存储桶的版本控制状态可以设置为以下的值：

- Enabled：对存储桶中的所有对象启用版本控制，之后每个添加到存储桶中的对象都会被设置一个唯一的version ID。
- Suspended：暂停存储桶的版本控制，之后每个添加到bucket中的对象的version ID会被设置为null。

代码示例

```
def put_bucket_versioning(self):
    resp = self.s3_client.put_bucket_versioning(
        Bucket='<your-bucket-name>',
        VersioningConfiguration={
            'Status': 'Enabled'      #'Enabled'|'Suspended'
        }, )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	存储桶的名称	是
VersioningConfiguration	VersioningConfiguration	封装了设置版本控制状态的参数	是

VersioningConfiguration说明

参数	类型	说明
Status	Status	Enabled|Suspended，版本控制开关状态

返回结果

根据返回码判断是否设置成功，200表示成功。

获取桶版本控制状态

功能说明

您可以通过get_bucket_versioning获取存储桶的版本控制状态信息。只有存储桶的拥有者才能获取到版本控制信息。如果存储桶从来没有被设置过版本控制状态，那么该存储桶不含有任何版本控制的状态信息，执行get_bucket_versioning操作不能获取版本控制信息的有效值。

代码示例

```
def get_bucket_versioning(self):
    resp = self.s3_client.get_bucket_versioning(
        Bucket='<your-bucket-name>',
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是

返回结果

参数	类型	说明
Status	Status	Enabled|Suspended，版本控制开关状态

对象相关接口

获取对象列表

功能说明

list_objects操作用于列出存储桶中的全部对象，该操作返回最多1000个对象信息，可以通过设置过滤条件来列出存储桶中符合特定条件的对象信息。

代码示例

```
def list_objects(self):
    print('list_objects')
    response = self.s3_client.list_objects(
        Bucket='<your-bucket-name>',
        MaxKeys=50,  # list up to 50 key at a time
    )
    for obj in response['Contents']:
        print(obj["Key"])
```

如果 list 大于1000，则返回的结果中 isTruncated 为true，通过Marker参数可以指定下次读取的起点。列举所有对象的示例代码如下：

```
def list_objects2(self):
    print('list_objects')
    objects = []
    response = self.s3_client.list_objects(
        Bucket=self.bucket,
        MaxKeys=100,
    )
    objects.extend(response['Contents'])
    while response['IsTruncated']:
        response = self.s3_client.list_objects(
```

```
            Bucket=self.bucket,
            MaxKeys=100,
            Marker=response['Contents'][-1]['Key']
        )
        objects.extend(response['Contents'])

    for obj in objects:
        print(obj["Key"])
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
MaxKeys	int	设置响应中返回的最大键数。默认值和可设置最大值均为1000	否
Prefix	string	指定列出对象的键名需要包含的前缀	否
Marker	string	用于在某一个具体的键名后列出对象，可指定存储桶中的任一个键名	否

返回结果

参数	类型	说明
Contents	Content数组	对象列表

上传对象

功能说明

您可以使用put_object接口直接上传文件。

代码示例

```
def put_object(self):
    print('put_object')
    key = '<your-object-key>'
    local_path = '<file-path>'
    with open(local_path, 'rb') as f:
        resp = self.s3_client.put_object(
            Bucket='<your-bucket-name>',
            Key=key,
            Body=f,
            # ACL='public-read',
            # ContentType='text/json',
            # StorageClass='STANDARD_IA',
        )
        print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名	是
Key	string	对象名	是
Body	bytes|file	要上传的数据，如打开的文件对象	是
ACL	string	对象访问权限，取值private | public-read | public-read-write	否
ContentType	string	http content-type header	否
StorageClass	string	配置上传对象的存储类型，包括标准类型STANDARD、低频类型STANDARD_IA以及归档类型GLACIER	否
Metadata	dict	自定义元数据	否

返回结果

参数	类型	说明
ETag	string	对象的唯一标签

注意：put_object对文件大小有限制，最大能上传5GB大小的文件，超过5GB需要使用分片上传。

下载对象

功能说明

您可以使用get_object下载对象。

代码示例

```
def get_object(self):
    print('get_object')
    key = '<your-object-key>'
    local_path = 'E:/ExampleObject.txt'
    resp = self.s3_client.get_object(
        Bucket='<your-bucket-name>',
        Key=key
    )
    body = resp["Body"]
    with open(local_path, 'wb') as f:
        for chunk in body:
            f.write(chunk)
```

请求参数

参数	类型	说明	是否必须
Bucket	string	桶名	是
Key	string	对象名	是

返回结果

参数	类型	说明
Body	StreamingBody	对象数据内容
Metadata	dict	自定义元数据

复制对象

功能说明

您可以使用copy_object复制一个已经在媒体存储中的对象。使用copy_object可以复制单个最大为5GB的对象。执行copy_object操作，必须具有对被拷贝对象的READ权限和对目标bucket的WRITE权限。

代码示例

```
def copy_object(self):
    key = '<dst-object-key>'
    resp = self.s3_client.copy_object(
        Bucket='<dst-bucket-name>',
        Key=key,
        CopySource={'Bucket': '<source-bucket-name>', 'Key': '<source-object-
key>'},
        ContentType="text/json",
        MetadataDirective='REPLACE',
        # StorageClass='STANDARD_IA',
    )
    print(resp)
```

文件比较大（超过1GB）的情况下，直接使用copyObject 可能会出现超时，需要使用分片复制的方式进行文件复制。TransferManager封装了分片复制的接口，可以用于复制文件。

```
class TransferDemo(object):
    def __init__(self):
        config = botocore.config.Config(
            signature_version='s3v4',  # s3 or s3v4
        )

        session = botocore.session.get_session()
        self.s3_client = session.create_client(
            's3',
            aws_access_key_id='<your-access-key>',
            aws_secret_access_key='<your-secret-key>',
            endpoint_url='<your-endpoint>',
            config=config)

        MB = 1024 * 1024
```

```
        transConfig = s3transfer.manager.TransferConfig(
            multipart_threshold=5 * MB,  # 大于该值使用分片上传
            multipart_chunksize=5 * MB,  # 分片大小
            max_request_concurrency=2,
            )

        # 设置带宽，不填表示不限制
        # transConfig.max_bandwidth = 1 * MB

        self.transfer = s3transfer.manager.TransferManager(self.s3_client,
transConfig)

    def copy(self):
        print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f'), "copy
start")
        source={'Bucket': '<source-bucket-name>', 'Key': '<source-object-key>'}
        dstBucket = '<dst-bucket-name>'
        dstKey = '<dst-object-key>'
        # 扩展配置
        extraArgs = {'ContentType': 'text/plain', 'ACL': 'public-read'}
        future = self.transfer.copy(source, dstBucket, dstKey,
extra_args=extraArgs)
        future.result()
        print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f'), "copy
success")
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	目的对象key	是
CopySource	CopySource	源对象	是
ContentType	string	设置目的对象的ContentType	否
StorageClass	string	配置目的对象的存储类型，包括标准类型STANDARD、低频类型STANDARD_IA以及归档类型GLACIER	否

关于CopySource：

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	源对象key	是

返回结果

参数	类型	说明
ETag	string	对象的唯一标签

删除对象

功能说明

您可以使用delete_object接口删除对象。

代码示例

```
def delete_object(self):
    key = '<your-object-key>'
    resp = self.s3_client.delete_object(
        Bucket='<your-bucket-name>',
        Key=key,
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名	是
Key	string	对象名	是

返回结果

根据返回码判断是否删除成功，204表示删除成功

批量删除对象

功能说明

您可以使用delete_objects接口批量删除多个对象，可以减少发起多个请求去删除大量对象的花销。delete_objects操作发起一个包含了最多1000个key的删除请求，媒体存储服务会对相应的对象逐个进行删除，并且将删除成功或者失败的结果通过response返回。如果请求删除的对象不存在，会返回已删除的结果。

delete_objects操作返回包含verbose 和quiet两种response模式。verbose response是默认的返回模式，该模式的返回结果包含了每个key的删除结果。quiet response返回模式返回的结果仅包含了删除失败的key，对于一个完全成功的删除操作，该返回模式不在相应消息体中返回任何信息。

代码示例

```
def delete_objects(self):
    resp = self.s3_client.delete_objects(
        Bucket='<your-bucket-name>',
        Delete={
            'Objects': [
                {
```

```
                    'Key': 'ExampleObject.txt',
                },
                {
                    'Key': 'ExampleObject1.txt',
                },
            ],
        },
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Delete	Delete	要删除的对象key列表	是

返回结果

参数	类型	说明
Deleted	数组	删除结果数组，包含每一个key的删除信息

获取对象元数据

功能说明

您可以使用head_object接口获取对象元数据信息

代码示例

```
def head_object():
    resp = s3_client.head_object(
        Bucket='<your-bucket-name>',
        Key='<your-object-key>',
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是

返回结果

参数	类型	说明
ETag	string	对象唯一标签
VersionId	string	对象的版本ID
ContentType	string	对象ContentType
Metadata	数组	自定义元数据
StorageClass	string	存储类型
ContentLength	int	对象大小

设置对象访问权限

功能说明

媒体存储支持一组预先定义的授权，称为Canned ACL。每个Canned ACL都有一组预定义的被授权者和权限，下表列出了相关的预定义授权含义。

ACL	权限	描述
private	私有读写	对象拥有者有读写权限，其他用户没有访问权限。
public-read	公共读私有写	对象拥有者有读写权限，其他用户只有该对象的读权限。
public-read-write	公共读写	所有用户都有该对象的读写权限。
authenticated-read	注册用户可读	对象拥有者有读写权限，注册用户具有该对象的读限。

您可以通过put_object_acl接口为媒体存储服务中的对象设置ACL。对一个对象执行该操作需要具有WRITE_ACP权限。

代码示例

```
def put_object_acl(self):
    key = '<your-object-key>'
    resp = self.s3_client.put_object_acl(
        Bucket='<your-bucket-name>',
        Key=key,
        ACL='public-read',
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是
ACL	string	acl值	是

返回结果

根据返回码判断是否设置成功，200表示成功。

获取对象访问权限

功能说明

您可以使用 get_object_acl操作获取对象的access control list（ACL）信息。

代码示例

```
def get_object_acl(self):
    key = '<your-object-key>'
    resp = self.s3_client.get_object_acl(
        Bucket='<your-bucket-name>',
        Key=key,
    )
    for grant in resp['Grants']:
        print(grant)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是

返回结果

参数	类型	说明
Owner	Owner	所有者信息
Grants	Grant数组	每种类型用户的详细权限信息

获取对象标签

功能说明

您可以使用get_object_tagging接口获取对象标签。

代码示例

```
def get_object_tagging():
    resp = s3_client.get_object_tagging(
        Bucket='<your-bucket-name>',
        Key='<your-object-key>'
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是
VersionId	string	设置标签信息的对象的版本Id	否

返回结果

参数	类型	说明
TagSet	TagSet	设置的标签信息，包含了一个Tag结构体的数组，每个Tag以Key-Value的形式说明了标签的内容

删除对象标签

功能说明

您可以使用delete_object_tagging接口删除对象标签。

代码示例

```
def delete_object_tagging():
    resp = s3_client.delete_object_tagging(
        Bucket='<your-bucket-name>',
        Key='<your-object-key>'
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	执行本操作的桶名称	是
Key	string	设置标签信息的对象key	是
VersionId	string	设置标签信息的对象的版本Id	否

返回结果

根据返回码判断是否删除成功，204表示删除成功

设置对象标签

功能说明

您可以使用put_object_tagging接口为对象设置标签。标签是一个键值对，每个对象最多可以有10个标签。bucket的拥有者默认拥有给bucket中的对象设置标签的权限，并且可以将权限授予其他用户。每次执行PutObjectTagging操作会覆盖对象已有的标签信息。每个对象最多可以设置10个标签，标签Key和Value区分大小写，并且Key不可重复。每个标签的Key长度不超过128字节，Value长度不超过256字节。SDK通过HTTP header的方式设置标签且标签中包含任意字符时，需要对标签的Key和Value做URL编码。设置对象标签信息不会更新对象的最新更改时间。

代码示例

```
def put_object_tagging():
    resp = s3_client.put_object_tagging(
        Bucket='<your-bucket-name>',
        Key='<your-object-key>',
        Tagging={
            'TagSet': [
                {
                    'Key': 'key1',
                    'Value': 'value1'
                },
            ]
        }
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是
Tagging	Tagging	设置的标签信息，包含了一个Tag结构体的数组，每个Tag以Key-Value的形式说明了标签的内容	是
VersionId	string	设置标签信息的对象的版本Id	否

返回结果

根据返回码判断是否设置成功，200表示设置成功

生成预签名URL

功能说明

您可以通过generate_presigned_url接口为一个指定对象生成一个预签名的链接，用于下载或上传对象。

代码示例

（1）生成一个预签名的下载链接：

```
def generate_getobject_presigned_url(self):
    print('generate_getobject_presigned_url')
    key = '<your-object-key>'
    url = self.s3_client.generate_presigned_url(
        ClientMethod='get_object',
        Params={'Bucket': '<your-bucket-name>', 'Key': key},
        ExpiresIn=900)
    print(url)
```

访问该预签名下载链接可以直接下载该对象：

```
def getObjUsingPresignedUrl(self, presigned_url, download_path):
    import requests
    response = requests.get(presigned_url)

    if response.status_code == 200:
        with open(download_path, 'wb') as file:
            file.write(response.content)
        print(f"Downloaded object saved to: {download_path}")
    else:
        print(f"Failed to download object. Status code: {response.status_code}")

    return response.status_code, response.text
```

（2）生成一个预签名的上传链接：

```
def generate_putobject_presigned_url(self):
    print('generate_putobject_presigned_url')
    key = '<your-object-key>'
    url = self.s3_client.generate_presigned_url(
        ClientMethod='put_object',
        Params={'Bucket': '<your-bucket-name>', 'Key': key},
        ExpiresIn=900)

    print(url)
```

通过该预签名上传链接可以直接上传该对象：

```
def putObjUsingPresignedUrl(self, presigned_url, file_path):
    import requests
    with open(file_path, 'rb') as file:
        response = requests.put(presigned_url, data=file)

    print(f"Upload Response Status: {response.status_code}")
    return response.status_code, response.text
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是
ExpiresIn	int	超时时间（秒）	否，默认3600秒

返回结果

参数	类型	说明
url	string	生成的链接

Post上传

功能说明

generate_presigned_post接口为一个指定对象生成一个支持post方式上传文件的参数集合，可以在前端使用post form-data的方式上传文件。

代码示例

```
def generate_postobject_presigned(self):
    print('generate_postobject_presigned')
    key = '<your-object-key>'
    Conditions = [
        ['starts-with', '$key', key],
    ]

    response = self.s3_client.generate_presigned_post(
        '<your-bucket-name>',
        key,
        Fields={},
        Conditions=Conditions,
        ExpiresIn=3600)

    print(response)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	bucket的名称	是
Key	string	对象的key	是
ExpiresIn	int	超时时间（秒）	否
Fields	数组	前端输入参数，用于配置acl，ContentType	否
Conditions	数组	参数策略，可以限制输入的参数	否

返回结果

参数	类型	说明
url	string	请求上传的url
key	string	对象的key
policy	string	服务端用于校验的policy
x-amz-algorithm	string	v4签名，哈希算法
x-amz-signature	string	v4签名，请求的参数签名
x-amz-date	string	v4签名，日期信息
x-amz-credential	string	v4签名，ak信息
AWSAccessKeyId	string	v2签名，ak信息
signature	string	v2签名，请求的参数签名

前端使用方式如下：

```
<form action="<data.url>" method="POST" enctype="multipart/form-data">
    <input type="hidden" name="Policy" value="<data.fields['Policy']>" />
    <input type="hidden" name="X-Amz-Algorithm" value="<data.fields['X-Amz-
Algorithm']>" />
    <input type="hidden" name="X-Amz-Credential" value="<data.fields['X-Amz-
Credential']>" />
    <input type="hidden" name="X-Amz-Date" value="<data.fields['X-Amz-Date']>"
/>
    <input type="hidden" name="X-Amz-Signature" value="<data.fields['X-Amz-
Signature']>" />
    <input type="hidden" name="bucket" value="<data.fields['bucket']>" />
    <input type="hidden" name="key" value="<data.fields['key']>" />

    <input type="file" name="file" value="" />
    <input type="submit" value="Submit" />
</form>
```

获取多版本对象列表

功能说明

如果桶开启了版本控制，您可以使用 list_object_versions接口列举对象的版本，每次list操作最多返回1000个对象。

代码示例

```
def list_object_versions(self):
    print('list_object_versions')
    response = self.s3_client.list_object_versions(
        Bucket='<your-bucket-name>',
        MaxKeys=50,  # list up to 50 key at a time
    )
    for obj in response['Versions']:
        print(obj["Key"])
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
MaxKeys	int	设置响应中返回的最大键数。默认值和可设置最大值均为1000	否
Prefix	string	指定列出对象的键名需要包含的前缀	否
Marker	string	用于在某一个具体的键名后列出对象，可指定存储桶中的任一个键名	否

返回结果

参数	类型	说明
Versions	数组	对象列表

分片上传接口

融合接口

功能说明

分片上传步骤较多，包括初始化、文件切段、各个分片上传、完成上传。为了简化分片上传，SDK提供了方便的融合接口用于上传文件，用户不需要关心文件的大小，SDK会自动对大文件使用分片上传。

代码示例

```
import botocore.config
import botocore.session
import s3transfer.manager
import datetime

class TransferDemo(object):
    def __init__(self):
        config = botocore.config.Config(
            signature_version='s3v4',  # 签名版本，s3 or s3v4
        )

        self.bucket = '<your-bucket-name>'
        session = botocore.session.get_session()
        self.s3_client = session.create_client(
            's3',
            aws_access_key_id='<your-access-key>',
            aws_secret_access_key='<your-secret-key>',
            endpoint_url='<your-endpoint>',
            config=config)

        MB = 1024 * 1024
```

```
        transConfig = s3transfer.manager.TransferConfig(
            multipart_threshold=5 * MB,  # 大于该值使用分片上传
            multipart_chunksize=5 * MB,  # 分片大小
            max_request_concurrency=2,
            )

        # 设置带宽，不填表示不限制
        # transConfig.max_bandwidth = 1 * MB

        self.transfer = s3transfer.manager.TransferManager(self.s3_client,
transConfig)

    def upload(self):
        print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f'), "upload
start")
        key = '<your-object-key>'

        # 扩展配置，可以设置ContentType和ACL
        extraArgs = {'ContentType': 'text/plain', 'ACL': 'public-read'}

        with open('<file-path>', 'rb') as f:
            future = self.transfer.upload(f, self.bucket, key,
extra_args=extraArgs)
            future.result()
```

请求参数

参数	类型	说明
bucket	string	桶名
key	string	对象名
fileobj	fileobj	打开的文件对象
extra_args	dict	扩展配置，可以设置ContentType和ACL；ACL取值private | public-read | public-read-write

TransferConfig参数，

参数	类型	说明
multipart_threshold	int	大于该值使用分片上传
multipart_chunksize	int	分片大小，默认5MB
max_request_concurrency	int	上传分片并发数

关于Content-Type的配置

Content-Type用于标识文件的资源类型，比如`image/png`，`image/jpg`是图片类型，`video/mpeg`，`video/mp4`是视频类型，`text/plain`，`text/html`是文本类型，浏览器针对不同的Content-Type会有不同的操作，比如图片类型可以预览，视频类型可以播放，文本类型可以直接打开。`application/octet-stream`类型会直接打开下载窗口。

有些用户反馈图片和视频无法预览的问题，主要就是Content-Type没有正确设置导致的；Content-Type参数需要用户主动设置，默认是`application/octet-stream`。在python sdk中，可以根据对象key值后缀扩展名来决定文件的Content-Type，参考代码如下：

```
import mimetypes

def mime_type(key):
    mt = mimetypes.guess_type(key)[0]
    if mt == None :
        return ""
    return mt
```

初始化分片上传任务

功能说明

您可以使用create_multipart_upload接口创建上传任务。

代码示例

```
# create_multipart_upload
resp = self.s3_client.create_multipart_upload(
    Bucket='<your-bucket-name>',
    Key='<your-object-key>'
)
upload_id = resp['UploadId']
print('create_multipart_upload success upload_id: %s' %upload_id)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是

返回结果

参数	类型	说明
UploadId	string	分片上传任务的id

上传分片

功能说明

初始化分片上传任务后，指定分片上传任务的id可以上传分片数据，可以将大文件分割成分片后上传，除了最后一个分片，每个分片的数据大小为5MB~5GB，每个分片上传任务最多上传10000个分片。您可以使用upload_part上传分片。

代码示例

```
def multipart_upload(self):
    bucket = '<your-bucket-name>'
    key = '<your-object-key>'
    local_path = '<file-path>'
    parts = []  # part list uploaded by client
    part_size = 5 * 1024 * 1024
    # create_multipart_upload
    resp = self.s3_client.create_multipart_upload(
        Bucket=bucket,
        Key=key
    )
    upload_id = resp['UploadId']
    print('create_multipart_upload success upload_id: %s' %upload_id)
    # upload_part
    with open(local_path, 'rb') as f:
        s = f.read(part_size)
        part_num = 1
        while s:
            file_chunk = io.BytesIO(s)
            resp = self.s3_client.upload_part(
                Bucket=bucket,
                Key=key,
                Body=file_chunk,
                UploadId=upload_id,
                PartNumber=part_num,
            )
            print('upload part %d success' %part_num)
            part = {
                'ETag': resp['ETag'],
                'PartNumber': part_num
            }
            parts.append(part)
            s = f.read(part_size)
            part_num += 1
    # complete_multipart_upload
    resp = self.s3_client.complete_multipart_upload(
        Bucket=bucket,
        Key=key,
        UploadId=upload_id,
        MultipartUpload={
            'Parts': parts
        },
    )
    print('complete_multipart_upload success upload_id: %s' %upload_id)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是
Body	bytes|file	对象的数据	是
PartNumber	int	当前分片号码	是
UploadId	string	通过创建上传任务接口获取到的任务Id	是

返回结果

参数	类型	说明
ETag	string	本次上传分片对应的Entity Tag

合并分片

功能说明

合并指定分片上传任务id对应任务中已上传的对象分片，使之成为一个完整的文件。您可以使用complete_multipart_upload接口合并分片。

代码示例

```
# complete_multipart_upload
resp = self.s3_client.complete_multipart_upload(
    Bucket='<your-bucket-name>',
    Key='<your-object-key>',
    UploadId=upload_id,
    MultipartUpload={
        'Parts': parts
    },
)
print('complete_multipart_upload success upload_id: %s' %upload_id)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是
MultipartUpload	MultipartUpload	包含了每个已上传的分片的ETag和PartNUmber等信息	是
UploadId	string	通过创建上传任务接口获取到的任务Id	是

返回结果

参数	类型	说明
ETag	string	本次上传对象后对应的Entity Tag

列举分片上传任务

功能说明

您可以使用list_multipart_uploads获取未完成的上传任务。

代码示例

```
def list_multipart_uploads(self):
    print('list_multipart_uploads')
    prefix = '<your-object-key>'
    resp = self.s3_client.list_multipart_uploads(
        Bucket='<your-bucket-name>',
        Prefix=prefix,
        MaxUploads=50,
    )
    for task in resp['Uploads']:
        print('key: %s, id: %s' %(task['Key'], task['UploadId']))
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
MaxUploads	int	用于指定获取任务的最大数量（1-1000）	否
Prefix	string	指定上传对象key的前缀	否

返回结果

参数	类型	说明
Uploads	Uploads	包含了零个或多个已初始化的上传分片信息的数组。数组中的每一项包含了分片初始化时间、分片上传操作发起者、对象key、对象拥有者、存储类型和UploadId等信息

列举已上传的分片

功能说明

您可以使用list_parts获取一个未完成的上传任务中已完成上传的分片信息。

代码示例

```
def list_parts(self):
    print('list_parts')
    key = '<your-object-key>'
    resp = self.s3_client.list_parts(
        Bucket='<your-bucket-name>',
        Key=key,
        UploadId='<upload id>',
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是
UploadId	string	指定返回该任务id所属的分片上传的分片信息	是

返回结果

参数	类型	说明
Parts	Parts	包含了已上传分片信息的数组，数组中的每一项包含了该分片的Entity tag、最后修改时间、PartNumber和大小等信息

复制分片

功能说明

复制分片操作可以从一个已存在的对象中拷贝指定分片的数据，当拷贝的对象大小超过5GB，必须使用复制分片操作完成对象的复制。除了最后一个分片外，每个拷贝分片的大小范围是[5MB，5GB]。在一个在拷贝大对象之前，需要使用初始化分片上传操作获取一个upload id，在完成拷贝操作之后，需要使用合并分片操作组装已拷贝的分片成为一个对象。您可以使用upload_part_copy复制一个分片。

代码示例

```
def multipart_copy(self):
    bucket = '<dst-bucket-name>'
    key = '<dst-object-key>'
    source_bucket = '<source-bucket-name>'
    source_key = '<source-object-key>'
    parts = []  # part list uploaded by client
    part_size = 5 * 1024 * 1024

    # head_object
    resp = self.s3_client.head_object(
        Bucket=bucket,
        Key=source_key
    )
    print('head_object success length: %s' %resp['ContentLength'])
    size = resp['ContentLength']
```

```
    # create_multipart_upload
    resp = self.s3_client.create_multipart_upload(
        Bucket=bucket,
        Key=key
    )
    upload_id = resp['UploadId']
    print('create_multipart_upload success upload_id: %s' %upload_id)
    # upload_part
    start = 0
    part_num = 1
    while start < size:
        if start + part_size < size:
            end = start + part_size - 1
        else:
            end = size - 1

        resp = self.s3_client.upload_part_copy(
            Bucket=bucket,
            Key=key,
            CopySource={'Bucket': source_bucket, 'Key': source_key},
            UploadId=upload_id,
            PartNumber=part_num,
            CopySourceRange='bytes=%d-%d' %(start, end)
        )
        print('copy part %d success' %part_num)
        part = {
            'ETag': resp['CopyPartResult']['ETag'],
            'PartNumber': part_num
        }
        parts.append(part)
        start = end + 1
        part_num += 1

    # complete_multipart_upload
    resp = self.s3_client.complete_multipart_upload(
        Bucket=bucket,
        Key=key,
        UploadId=upload_id,
        MultipartUpload={
            'Parts': parts
        },
    )
    print('complete_multipart_upload success upload_id: %s' %upload_id)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	目的对象key	是
PartNumber	int	当前分片号码	是
UploadId	string	通过创建上传任务接口获取到的任务Id	是

UploadId	string	通过创建上传任务接口获取到的任务id	是
CopySource	string	源对象	是
CopySourceRange	string	指定本次分片拷贝的数据范围，必须是"bytes=first-last"的格式，例如"bytes=0-9"表示拷贝原对象中前10字节的数据，只有当拷贝的分片大小大于5MB的时候有效	否

| 参数 | 类型 | 说明 | 是否必要 |

关于CopySource：

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	源对象key	是

返回结果

参数	类型	说明
CopyPartResult	CopyPartResult	包含拷贝分片的Entity Tag和最后修改时间等信息

取消分片上传任务

功能说明

您可以使用abort_multipart_upload终止一个分片上传任务。

代码示例

```
def abort_multipart_upload(self):
    print('abort_multipart_upload')
    key = '<your-object-key>'
    upload_id = '<upload-id>'
    resp = self.s3_client.abort_multipart_upload(
        Bucket='<your-bucket-name>',
        Key=key,
        UploadId=upload_id,
    )
    print(resp)
```

请求参数

参数	类型	说明	是否必要
Bucket	string	桶名称	是
Key	string	对象key	是
UploadId	string	需要终止的上传任务id	是

安全凭证服务（STS）

STS即Secure Token Service 是一种安全凭证服务，可以使用STS来完成对于临时用户的访问授权。对于跨用户短期访问对象存储资源时，可以使用STS服务。这样就不需要透露主账号AK/SK，只需要生成一个短期访问凭证给需要的用户使用即可，避免主账号AK/SK泄露带来的安全风险。

初始化STS服务

```
access_key = '<your-access-key>'
secret_key = '<your-secret-access-key>'
end_point = '<your-endpoint>'
region = 'cn'

self.sts_client = boto3.client(
    'sts',
    aws_access_key_id=access_key,
    aws_secret_access_key=secret_key,
    endpoint_url=end_point,
    region_name=region)
```

获取临时token

```
def assume_role(self):
    print('assume_role')
    bucket = '<your-bucket>'
    policy = r'{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":
["s3:*"]' \
             r',"Resource":["arn:aws:s3:::%s","arn:aws:s3:::%s/*"]}}' % (bucket,
bucket)
    role_arn = "arn:aws:iam:::role/<your-role>"
    role_session_name = "<your-session-name>"

    print('policy: %s' % policy)
    response = self.sts_client.assume_role(
        Policy=policy,
        RoleArn=role_arn,
        RoleSessionName=role_session_name,
    )
    print('ak %s' % response['Credentials']['AccessKeyId'])
    print('sk %s' % response['Credentials']['SecretAccessKey'])
    print('token %s' % response['Credentials']['SessionToken'])
```

Policy设置例子

```
允许所有的操作
{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":
["s3:*"],"Resource":["arn:aws:s3:::<your-bucket-name>","arn:aws:s3:::<your-
bucket-name>/*"]}}

限制只能上传和下载
{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":
["s3:PutObject","s3:GetObject"],"Resource":["arn:aws:s3:::<your-bucket-
name>","arn:aws:s3:::<your-bucket-name>/*"]}}

使用分片上传
```

```
{"Version":"2012-10-17","Statement":{"Effect":"Allow","Action":
["s3:PutObject","s3:AbortMultipartUpload","s3:ListBucketMultipartUploads","s3:Li
stMultipartUploadParts"],"Resource":["arn:aws:s3:::<your-bucket-
name>","arn:aws:s3:::<your-bucket-name>/*"]}}
```

其他操作权限
上传权限：s3:PutObject
下载权限：s3:GetObject
删除权限：s3:DeleteObject
获取列表权限：s3:ListBucket

注意：
1.ListObjects 操作是由ListBucket权限控制的
2."Version:2012-10-17"是系统的policy格式的版本号，不能改成其他日期

更多操作权限可以参考：
https://www.ctyun.cn/document/10306929/10136179

参数	类型	描述	是否必要
RoleArn	String	角色的ARN，在控制台创建角色后可以查看	是
Policy	String	角色的policy，需要是json格式，限制长度1~2048	是
RoleSessionName	String	角色会话名称，此字段为用户自定义，限制长度2~64	是
DurationSeconds	Integer	会话有效期时间，默认为3600s，范围15分钟至12小时	否